Introduction

Optical character recognition, the ability to recognize an image of text and convert it to digital format, is a widely studied and well understood. By applying similar principles, symbols on a sheet of music, can be processed and easily converted to digital form.

As motivation for our research, we imagine our algorithm as a framework for a smartphone application, which complements a musician who needs a rendition of a song as a paradigm to emulate. An image taken by the musician is processed and subsequently synthesized into an instrumental piece.

With no similar application for music notation recognition available in the smartphone market, the creation of such a framework could have a potentially revolutionary impact. We will demonstrate, as a proof a concept, signal processing techniques, such a matched filters, and software strategies, of an object-oriented implementation, for designing such a framework in Matlab.

Overview

Design Assumptions

Due to the complexity and proliferation of musical notations, our music notation analysis focuses on a few elementary and widely used musical symbols, providing the foundation for further research.

- Perfect Image Orientation: The music notation analysis assumes a preprocessing stage that scales an image of sheet music and corrects orientation.
- Time Signature Independence: Music speed is dependent on both time-signature (e.g. 4-4 time) and temp (e.g. Moderato). Current algorithm implementation assumes a Moderato, 4-4 time, with 120 beats-per-minute.
- Key Signature Independence: The program assumes a preprocessing stage that determines key signature, which then shifts note values by +/- a half note frequency. In this design we assumed the C-major.
- Treble Clef: The program assumes a preprocessing stage that determined the clef as treble clef.
- Common Notations: Notations matches occur with whole, half, quarter, eighth, rests, sharps, and flats.

Cross-Correlation Matched Filter Algorithm:

- Filters are used to remove unwanted frequency components from a signal, in order to sift out components of interest. In this case we want to determine what segments of a sheet of music have similar frequency content to that of an image of a certain note. In signal processing, a matched filter detects the presence of a known signal in a template signal. Matched filtering is widely used in communications for determining the presence of one signal or another (e.g. the representation of a one vs the representation of a zero in transferring bits). In this case we want to determine what kinds of notes are present in our image and where they are located.
- There are a number of ways to do matched filtering, one of which is to compute the cross-correlation between the signal you’re looking for

and a template signal (the signal in which you're searching for your signal of interest). Performing a cross-correlation is essentially the same as performing a convolution, but without first "flipping" one of the signals:

Where y[n] denotes the matched output, h[n] denotes template image, and x[n] denotes the image.

- Performing the 2D cross correlation of an inverted, binarized image (black pixels=1, white pixels=0, with pixels converted to either one or the other by thresholding) of a certain note with the inverted, binarized image of a segment of sheet music yields a matrix wherein the maximum entries correspond to locations on the template image where the similarity between the image of the note and that section of the template image (of the same dimensions) is at a maximum. Hence, by filtering with different note images (i.e. half, quarter, whole) we can determine not only the location of each note relative to the bars (and therefore it's type: a, b, c, etc.) but also the length of the note (half, quarter, whole, etc.).

Detailed Procedures

Object-Oriented Design

Each note has a set of information relevant to it, such as pitch, octave, frequency, etc. We use object-oriented design to represent a note, with the following parameters:

Property	Description
Vertical Position	Position from top of the page.
Horizontal Position	Position from left of the page.
Type	Whole, half, quarter, eighth notes.
Pitch	Note, i.e. A, B, C, D
Octave	Octave in relationship to fifth octave.
Frequency	Frequency in Hz.
Length	Time duration in seconds.
Accidental	Whether a sharp or flat is present.

Note class properties

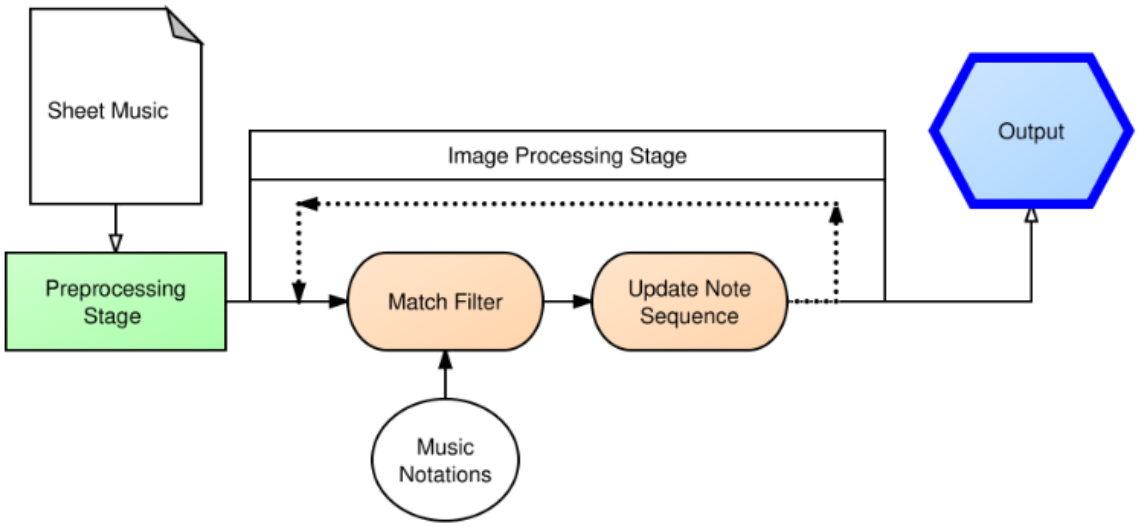
Above all else, implementing an object-oriented design gives room for future development. For example, lines in a stanza, clefs, key signatures, and accidentals can all be abstracted to objects. This will allow modification to the original music, such as changing the key signature,

adding or removing notes.
This means that:
The music can be edited to be drastically different from the original.
The music can be output not just as sound, but also to a newly edited sheet of music.

Tolerance Levels

Because of noise, note symbols required a threshold level to approximate the match. A symbol was matched if it was over 0.9 of the expected value. Due to the different shapes of notes, our program was tailored to have different tolerance levels for specific notes. These tolerance levels were set by determining the highest tolerance level that was able to discover all notes for a variety of test cases. These test cases involved placing a single note on all line positions. Pre-processing stages, such as removing inverting the pixel values of stanza lines, improved tolerance requirements.

System Flow



System Design Flow

System Flow

- **a. Load Image:** An image of sheet music is loaded and pixels are converted to 256-bit values.
- **b. Preprocessing Stage:**
    - The image is normalized to black and white pixels. Pixels appearing more black are give a value of 1 and white pixels are given a value of 0.
    - To determine a note's pitch, the location relative to the line position is required. The line position of a stanza (5 lines) of music is determined by averaging pixel values across the horizontal direction. Because an individual line may incorporate multiple pixel values in the vertical direction, the stanza line positions are determined by groups of nearest neighbors. The line positions are determined by taking the average point in horizontal direction. In our implementation, the value of the white pixel, w = 0, and the black pixel, b = 1. Therefore, the average value can be found with the following formula: ——
      —— , Where represents the location of a stanza.
    - Removing the stanzas from the image makes it easier to match individual notes. Black pixels are converted to white pixels along the stanza line locations, when a musical symbol is not present. A stanza section is not converted to a white pixel if a black pixel exits at the Average line width +/-1 from the line center.
- **c. Image Processing Stage:**
    - Matched Filter Cross-Correlation: In this case we want to determine what segments of a sheet of music have similar frequency content to that of an image of a certain note. Performing the 2D cross correlation of an inverted, binarized image (black pixels=1, white pixels=0, with pixels converted to either one or the other by thresholding) of a certain note with the

inverted, binarized image of a segment of sheet music yields a matrix wherein the maximum entries correspond to locations on the template image where the similarity between the image of the note and that section of the template image (of the same dimensions) is at a maximum. Hence, by filtering with different note images (i.e. half, quarter, whole) we can determine not only the location of each note relative to the bars (and therefore it's type: a, b, c, etc.) but also the length of the note (half, quarter, whole, etc.). Because of different standardizations in notations, each note type is given its own unique tolerance level, subsequently providing clusters denoting the peaks of the cross-correlation match with an individual note template.

Matched Filter Template for Eight Note

- Redundancy Checker: The correlation cluster locations, provided by the matched filter output, are averaged to find the center of the cluster. Before finding the cluster centers, the cluster groups must be found. A starting point, signifying the start of a new cluster, is created from the list of possible locations. This region is expanded by looking at nearest neighbors with a specific distance, approximately 1/4 the distance between two line stanzas, from the edge of the cluster. The cluster continues to expand if points from the possible locations are within the specified distance of the

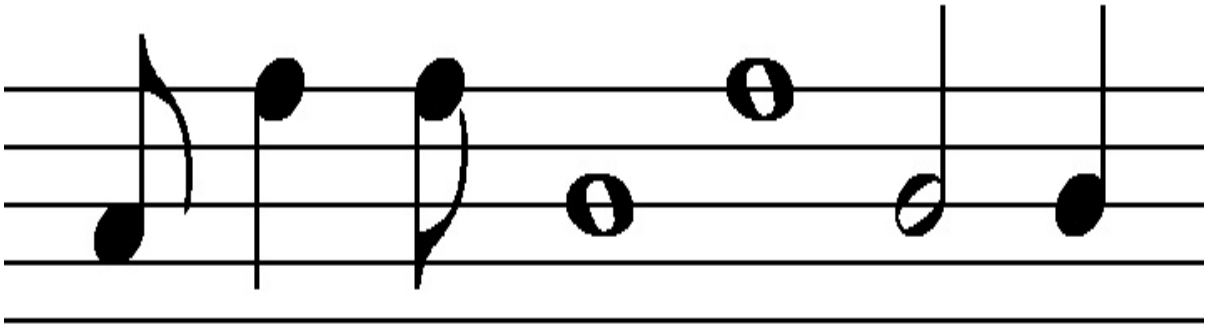
edges of the cluster. If no new points can be added to the cluster region, the center of this cluster region is determined by averaging the x and y positions, and the redundancy checker continues to check the remaining possible locations.

- **d. Output Tune:** The notes and other types of objects are concatenated to form a list, which is then sent to a music synthesizer program. Synthesis of the musical output from sheet music is complemented by the Wind Instruments Synthesis Toolbox, complements of El Instituto de Ingeniería Eléctrica. This toolbox synthesizes a variety of instruments for specific lengths and frequencies.

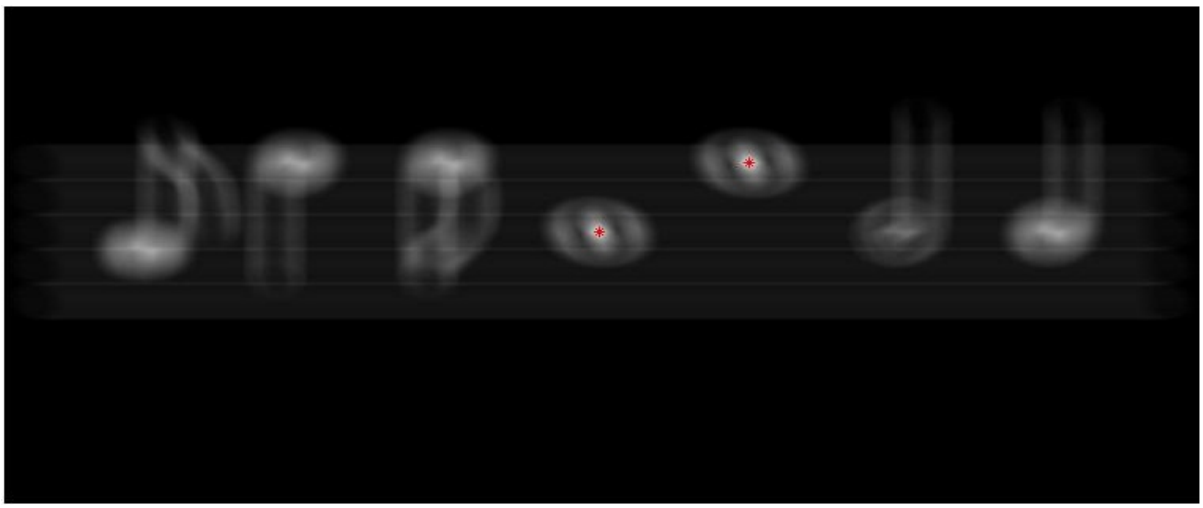
Results

Matched-Filtering

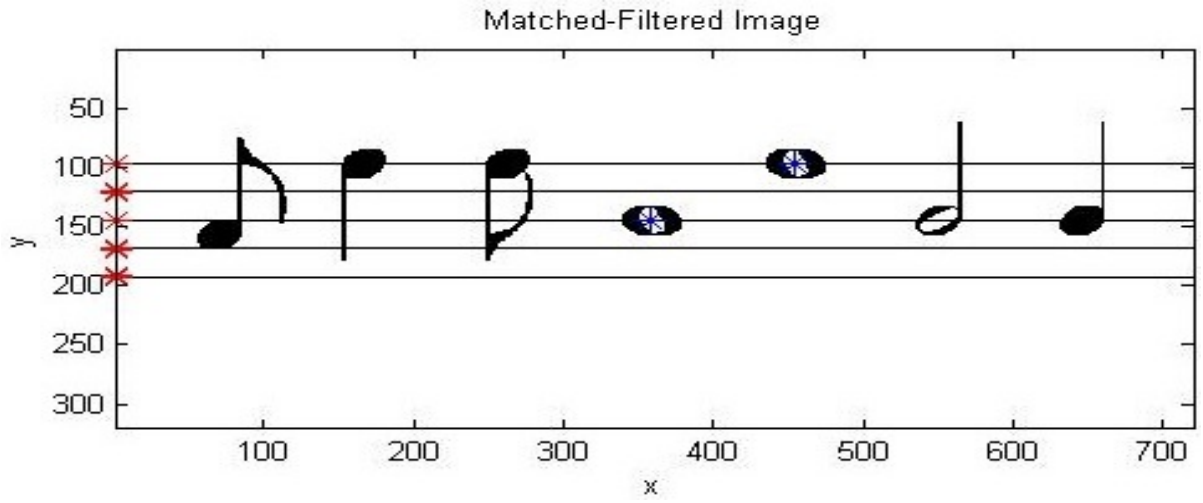
Our program was tailored to work flawlessly on our self-generated sheet music. Under our provided boundary conditions, all music notes were correctly identified and converted into the data parameters of length, octave, and frequency.
The first picture is a self-generated unprocessed sheet of music that we intend to analyze:



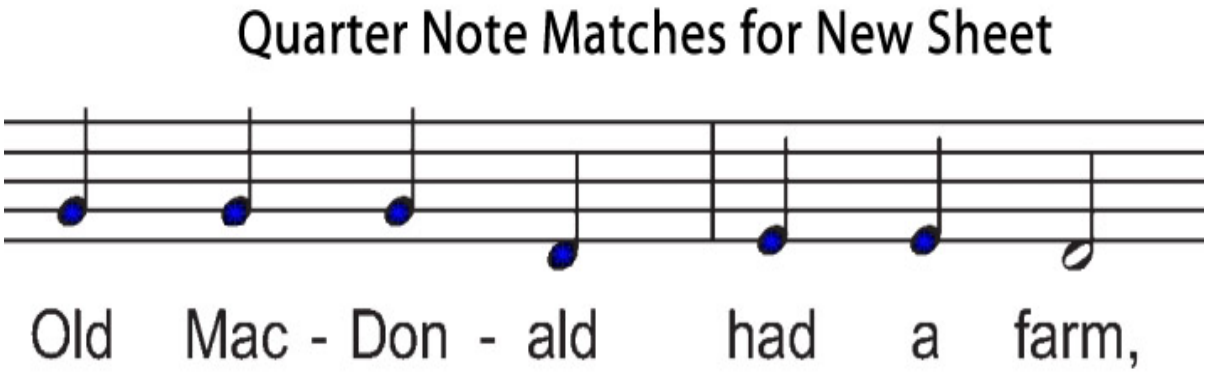
The second picture shows the correlation gradient after applying a whole notes matched filter. The red X's indicate where the hits are:



Finally, the third picture shows the hits back on the original sheet of music:



We also tried our program on a sheet of sheet music that we scan into the computer. Before running our program on the image we made sure the staffs were oriented correctly and that the image was close to the same scale as our self-generated pictures. The following picture shows that our program shows hits for quarter notes accurately:



As a note to the reader, our program has trouble recognizing eighth-note tails, but they were found when the tolerance level was lowered.

Conclusion

The ability to provide notation recognition for sheet music appears to be a very manageable and straightforward problem due to the common standardizations in musical notation. Our demonstration show that as a proof of concept, music interpretation and synthesis can be automated by a computer with high accuracy.

By implementing an object oriented design, our program provides a scalable solution to music notation recognition. Our program creates matches for different notation types and can be easily expanded by adding new notation templates with minor modifications to the underlying source code. After music notations are recognized, the applicable parameters provide a framework for transforming the the digitized music into a multitude of applications, such as a synthesized audio file, back to sheet music in a new scale and time signature, or lay the foundations for applications involving analysis of music theory.

Future Work

Immediate work would be to expand our notation recognition to new music types. While our program is able to detect sharps and flats and our OOP design created modules for these parameters, we have yet to connect the frequency shifts in our code. In terms of program flow, recognizing clefs and key signatures, which can shift the frequencies of the applicable lines, appears to be the next step to provide analysis for the basic sheet music. After this step, the next area of research would be towards accurately recognizing different music notation styles. This might include changing tolerance levels, creating a more generic match template, and perhaps adding another layer of image processing, such as detection of white spaces. In addition, a preprocessing stage that first extracted all music notations, without awareness of type, could improve the speed of the program.

As the algorithm expands, we recognizing the emphasis may shift towards creating a priority queue and the best framework. For example, in what

order is tempo, key signature, and notes scanned and processed. If two notes appear to have a match, which note type receives priority. These design strategies can have an impact on speed and performance, and play a role should this program be implemented on a mobile device, such as an iPhone.

Note to the reader:
While working on this project, we discovered music instrument synthesis is becoming very vibrant and fluid. As a thought experiment, we wonder the possibility of a single musical note being represented by 100 parameters, of 1 byte each. If a 3 minute song has approximately 1000 notes, a song could be processed and stored as a 10 kilobyte file.

References

McGee, Ryan. "Simple Music in MATLAB."
http://www.lifeorange.com/MATLAB/MATLAB_music.htm

"Making Music with MATLAB." Fall 1997, 2nd Quarter
http://users.rowan.edu/~shreek/networks1/music.html

"Wind Instruments Synthesis Toolbox"
http://iie.fing.edu.uy/~rocamora/wind_synthesis/doc/

Meet the Team

Team Members

Kevin Ting (kht2@), a Signal and Systems ECE student, from Hanszen College.

Richard Latimer (rpl1@), a Computer Engineering ECE student, from McMurtry College.

Kevin Beale (kdb2@), a Signal and Systems ECE student, from McMurtry College.

George Chen (mc20@), a Computer Engineering ECE student, from McMurtry College.

Acknowledgements

We would like to thank Chinmay Hegde for guiding us through this project and Dr. Baraniuk for giving us the opportunity to learn about this topic.